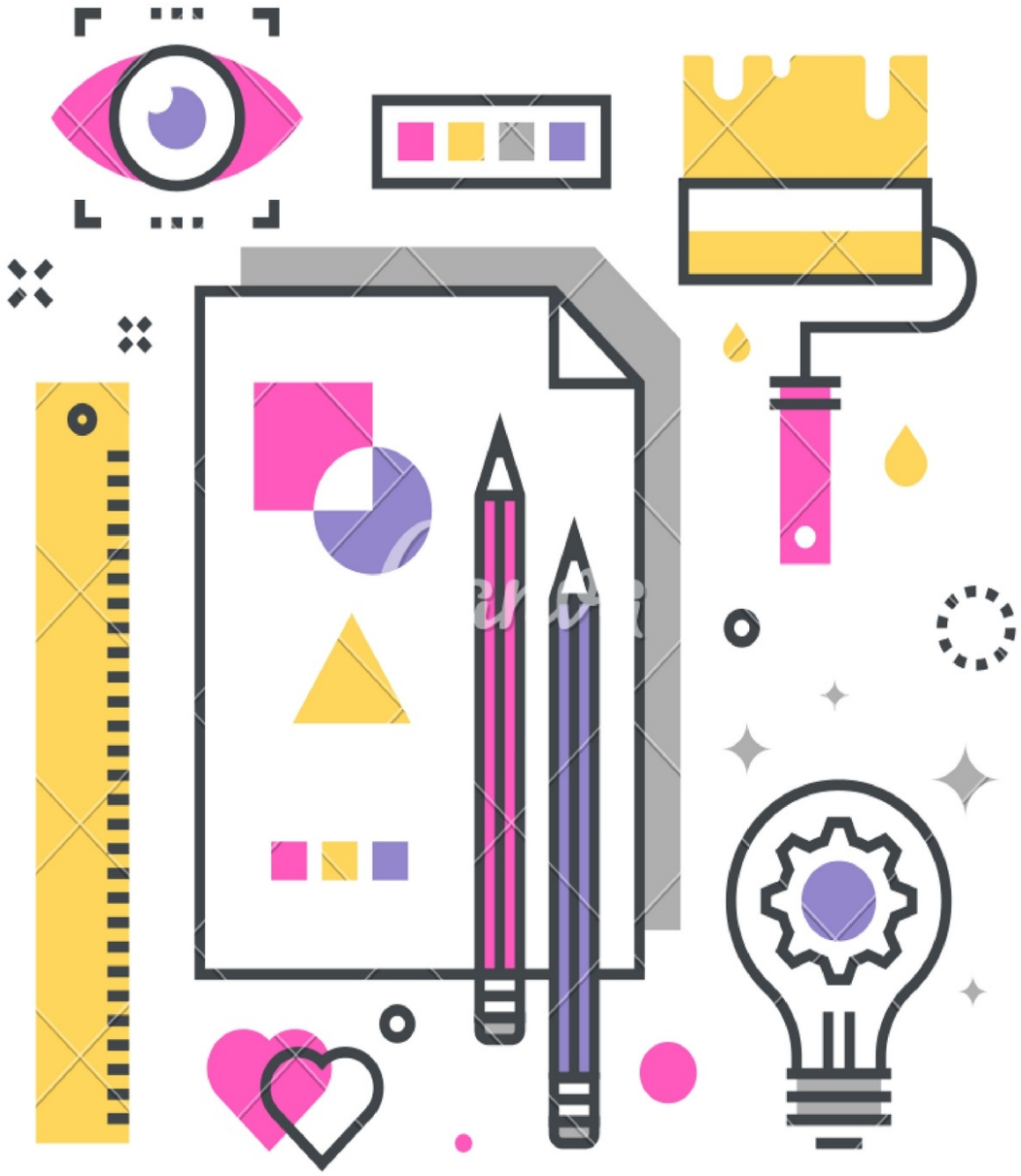
一份开源的IC 读物

了解CAD 知识， 了解行业大牛， 了解IC 方法论

IC GEEK

IC 极客



BY ICXHUB

Table of Contents

IC 极客

技术服务支持：support@icxhub.com

产品购洽联系：sale@icxhub.com

1. 成书使用的开源工具

- Dia - 流程图 (*.dia)
- GIMP - 图像编辑 (*.xcf)
- Gitbook - 内容组织
- Calibre - PDF 生成
- Node.js - gitbook 依赖
- VSCode - 文本编辑

开刊词

1.2.1 - 4WAIVGUZ

“IC极客”专栏开栏了，特邀您入圈。这是Alice和她的小伙伴们共同发起的专栏，我是Alice，IC行业的一名CAD老兵，。在本栏，“IC极客”不特指IC世界里佝偻在电脑前深宅的程序员，我们将它定义为一种对IC 技术的好奇心和让自己、团队乃至整个行业变得更好的执行力。接下6个月，在“IC极客”专栏你会看到丰富、夯实、专业的内容。

1. | AI 已来，如何应对人类学习的焦虑

2016年阿老师战胜世界顶级围棋高手李世石，机器学习正式走进大众视野，引起了人们极大的好奇及『讨论』上的恐慌。2017年，国内半导体行业如火如荼，上有国家战略和政策扶持，下有各路金主爸爸强力进入，Startup公司如雨后大蘑菇，成片冒出。魏老师说AI无疑是当前社会最关注的热点，不管有什么好的AI算法，要想最终得到应用，就必然要通过芯片来实现。未来已来，无论好坏。

当前世界在快轨上狂奔，基于新技术的新机遇层出不穷，却又稍纵即逝，只有那些跟得上时代步伐且掌握新技术的人才能抓住这些机遇。如何跟上世界狂奔的步伐，如何及时掌握最新的技术，如何抓住稍纵即逝的机遇，这些大概就是焦虑的来源。“IC极客”专栏将试着去探讨：IC工程师在各自的工作岗位上需要的学习能力及如何对抗随之而来的焦虑，试试看能不能找到一种行之有效的方法。

2. | 连接进IC 极客圈的思维矩阵

曾被合伙人问及：Alice你刚工作的时候，公司给你们培训什么？我想了想，那时公司有内部的e-learning 系统，里面有工具、工艺、设计、编程等各个领域的培训资料，可以自己进去看，讲师都是公司的一线大咖。与此同时在具体工作中每个人都有自己的mentor。

现在想来这样的系统也有其弊端，第一，该培训系统致力于把人培养成细分领域的操作工，而不负责培养某种学习能力，也不负责教授面向行业通用的领域知识；第二，不是每个mentor，都具备做mentor的素养和技术管理能力；第三，很多人在自己职业成长的路上看不到Roadmap。前两天有个刚入行的孩子跑来问我，学姐，我想做到像你一样，有什么方法？这个问题着实不好回答，但那孩子还是很执拗地问，每个阶段需要学习什么内容、达到什么目标、用什么方法、有没有一个guideline？对于这样大而有的问题，简单回答基本没有什么实质帮助。

所以，在这次专栏内容的制作中，我们会找一些真正的IC极客，用六个月的时间来共同探讨这些问题。我们将试图创建一个“IC极客圈”，聚集一群以极客精神对IC设计的细分领域或者整个行业有深刻好奇心和行动力的人。他们是来自IC 设计、前端、后端、验证、模拟等各个领域的资深工程师、技术经理或某位行业大咖。文章内容将以多种方式呈现，希望IC极客们的思维方式方法可以给大家在未来持续学习过程中带来一些启发。

我们期待富有极客精神的你愿意把自己连接进这个思维矩阵，大家一起行动。本着极客圈『开源和分享』的精神内核，整个IC极客专栏的文字及音视频内容将全部以开源、共享的形态呈现给大家。

3. | 未来6个月你将得到

在未来6个月里，我们会以编程@IC设计，IC极客的技术雷达，流程构建的核心能力，技术管理的核心能力，IC设计中观、微观与宏观为主线，探讨一些通用和细分领域的知识、技能及方法，分享对相关领域的问题思考、思维模型和方法论。为了夯实，会引入工程实践中的具体问题、具体案例及具体的解决之方。这不是一个完整的课程体系，每个模块，都是每个硅农在职业成长中或早或晚会遇到的通识性问题集结。

1. 编程@IC设计

这个模块主要由几位资深的CAD操刀，梳理编程思想以及不同的编程语言在IC设计各个领域的应用和最佳实践。时下最热门的计算机思维、大数据概念、最热的编程语言都会涵盖其中。

1. IC极客的技术雷达

这个模块由IC 设计各个领域的专家，特别是技术管理者，分享工程师在职业成长中的技术雷达和升级打怪指南。也会对每个领域的技术趋势做深入探讨。

1. 流程构建的核心能力

在国内流程构建主要是由领域专家和CAD共同完成，在不同企业里有不同的策略。完成流程构建需要关注哪些核心能力、需要采用何种技术方、如何实现弹性设计？就这些问题我们将做深入探讨。

1. 技术管理的核心能力

成为技术管理者是大部分工程师职业规划中的一个『小目标』，怎样从职场小白攀爬到管理层？技术管理者需要具备怎样的核心能力？作为工程师如何向上管理？就此，希望可以提供一些思维模型和行动指南助你事半功倍。

1. IC设计中观、微观与宏观

抱着flow 刷脚本，一般出不了什么大乱子，不按规定来，芯片容易做成砖。这里的Flow 可以理解为套路，是一个中观的概念。但是，抱着Flow刷脚本是远远不够的，你需要对每个不同的case有微观认知，这才是长在你身上的know-how，是flow不可替代的。宏观是更高一级的抽象，作为一名IC工程师需要具备zoom in和zoom out的能力，要既能看到宏观又能深入微观，这才算具备了高效定位和解决问题的能力。这一模块会探讨如何在升级打怪中一步步地练出这个能力。

期待在这里和你相遇！ Alice

文章中所提到到的延伸阅读、代码分享、后续更新通知及音视频上线通知都会发在IC极客交流群，同时，IC极客圈的小伙伴们也会在群里和大家一起交流学习。

欢迎扫码进群:

IC 极客交流群

该二维码7天内(4月24日前)有效，重新进入将更新

1.2.2 - 扫码进群

或加sgsphoto 好友，拉你进群:

1.2.3 - 加好友

IC 为何偏恋三十而立的Tcl

1.3.1.1 - 52HS9DF0

本文经「原本」原创认证，访问yuanben.io查询【52HS9DF0】获取授权信息。

Tcl，发音做"tickle"，是Tool Command Language的首字母缩写。

有些编程语言单独看起来并没有优势，脱离开应用场景后甚至可以说是一门即将被淘汰的语言，这一篇我们来看看Tcl 没有被淘汰、没有被取代的原因在哪里（我认为单独来看Tcl优势很弱）。

2018，Tcl 诞生已经三十个年头了，它依然做为IC 设计的主力语言存在着，为什么IC 偏偏恋上了Tcl，本文就Tcl 这门语言的特点谈谈我对这个话题的看法，用开放的心态聊聊Tcl。

如果你希望了解更多程序语言发明大牛对Tcl 和其他一些程序有趣的论战，可以去搜索 “Tcl War”。

- 第一部分有一个**Tcl** 的小测验，看看你可以得几分，了解你和**Tcl** 的亲密度。
  - 槽点：要不要评论回复下你的分数 :)
- 第二部分细数了**Tcl** 的优点和不足，快而立之年的**Tcl** 为何依然在**IC** 中发光发亮。
  - 看点：了解Tcl 语言产生的背景、历史与演变，或许可以给我们一点启发，我们为什么还在用它。
- 第三部分我会介绍一些我的经验，我心目中好的**IC Tcl** 程序的特点以及推荐两个好的学习实例。
  - 看点：一点学习Tcl 的体会，即使你厌恶编程，为了提高效率，为了绩效，为了工资，为了面试，你应该看一眼。
  - 亮点：只用Tcl、Tk、Expect 也可以创造一个自己的IC 黑客帝国。

1. 测一测你对**Tcl** 的了解

如果不知道或者回答否则不得分，回答是或者知道得1分。6-8分及以上说明你对Tcl 的了解还是不错的。

- 不管你得几分，不管你是不是喜欢**Tcl -** 这篇文章都可以给你一些故事和建议。
- 多家EDA 工具的Tcl Shell和Linux/Unix 上原生Tcl Shell，你（觉得）哪个好用？
- 你是否（熟练使用）下面全部的命令（1）？
  - set
  - puts
  - array
  - source
  - if/switch
  - for/foreach
- 你是否（熟练使用）下面全部的命令（2）？
  - open/close
  - file
  - regexp/regsub
  - proc
  - clock

- 你是否(写过)procedure？
- 你(知道)2个以上Tcl 中作用域相关的命令吗？
- 你是否(用过)Pakcage, 标准库的Package或者其他人写的Package？
- 你是否(写过)Package, 并知道它如何被使用？
- 你(用过)dict 吗？
- 你(知道)namespace 和它的用法吗？
- 你(用过)interp 这个命令吗？
- 你(知道)Tcl 如何Trace 吗？
- 你(知道)有个TK 库Bwidget吗？
- 你是否(知道)如何获取http 数据？
- 你是否(用过)wait 和send 相关的命令？
- 你(编写)过C 语言功能并在tcl 中load 吗？

2. Tcl 的诞生历史

看点:了解**Tcl** 语言产生的背景、历史与演变, 或许可以给我们一点启发, 我们为什么还在用它。

Tcl/Tk 的发明人 John Ousterhout 教授在八十年代初, 是伯克利大学的教授。在其教学过程中, 他发现在集成电路 CAD 设计中, 很多时间是花在编程建立测试环境上。并且环境一旦发生了变化, 就要重新修改代码以适应。这种费力而又低效的方法, 迫使 Ousterhout 教授力图寻找一种新的编程语言, 它即要有好的代码可重用性, 又要简单易学, 这样就促成了 Tcl (Tool Command Language) 语言的产生。

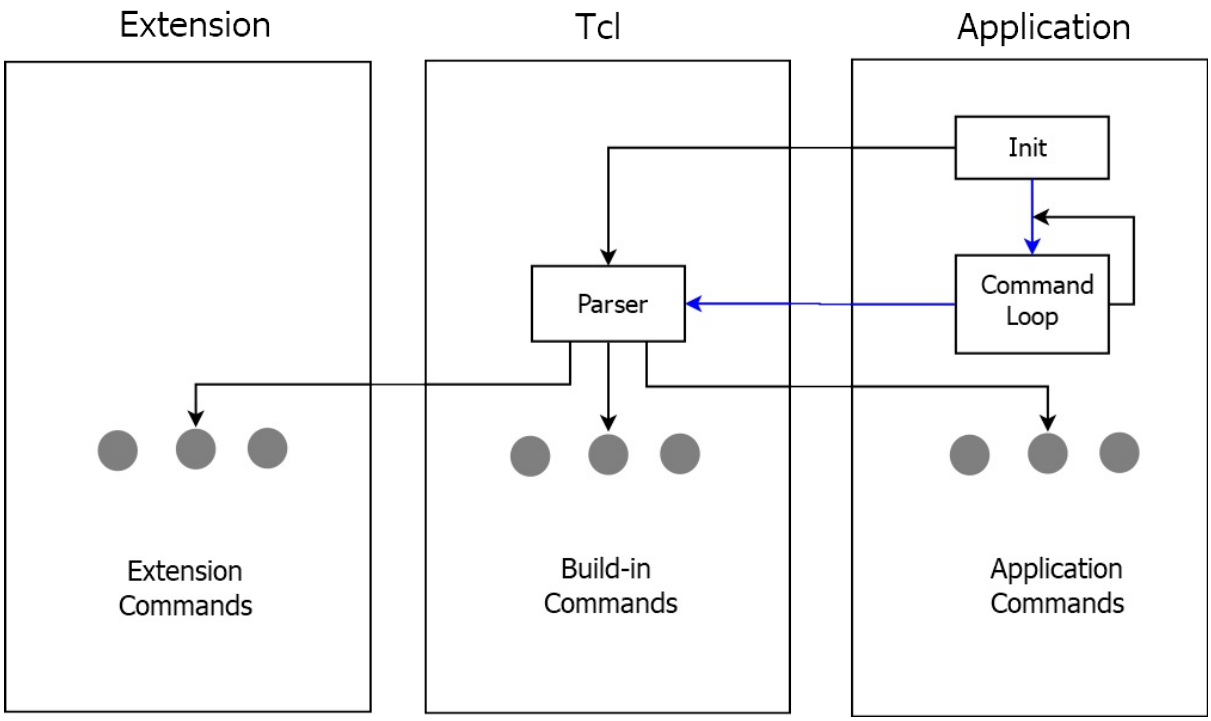
Ousterhout 教授被 SUN 于 1994 年召致麾下, 加入 SUNLab, 领导一个小组从事将 Tcl 移植到所有其它平台的工作, 如 Windows 和 Macintosh。同时为 Tcl 增加了 Safe-Tcl 安全模块, 并为浏览器的开发了 Tcl plug-in, 以及支持 Java bytecode 的编译器, 大字符集, 新的 IO 接口, 与 Java 的计算平台相连等。Tcl还有自己的浏览器。在面向对象程序设计占主导地位的今天, 又开发了支持面向对象的 incr Tcl。为了鼓励各厂商开发第三方的程序, Tcl 的源代码可免费下载。所有这些努力使 Tcl 成为一个适应当代信息产业潮流的, 支持多平台的, 优秀的的开发语言。

Ousterhout 教授于 1998 年初, 离开了 SUN, 自立 Scriptics 公司, 继续 Tcl/Tk 的研究和开发工作。

Tcl 最初的构想的是希望把编程按照基于组件的方法 (component approach), 即与其为单个的应用程序编写成百上千行的程序代码, 不如寻找一个种方法将程序分割成一个个小的, 具备一定“完整”功能的, 可重复使用的组件。这些小的组件小到可以基本满足一些独立的应用程序的需求, 其它部分可由这些小的组件功能基础上生成。不同的组件有不同的功能, 用于不同的目的。并可为其它的应用程序所利用。

当然, 这种语言还要有良好的扩展性, 以便用户为其增添新的功能模块。最后, 需要用一种强的, 灵活的“胶水”把这些组件“粘”合在一起, 使各个组件之间可互相“通信”, 协同工作。程序设计有如拼图游戏一样, 这种设计思想与后来的 Java 不谋而合。终于在 **1988** 年的春天, 这种强大灵活的胶水 **- Tcl** 语言被发明出来了。

按照Ousterhout 教授的定义, **Tcl** 是一种可嵌入的命令脚本化语言 **(Command Script Language)**。“可嵌入”是指把很多应用有效, 无缝地集成在一起。“命令”是指每一条 Tcl 语句都可以理解成命令加参数的形式:



1.3.1.2 - Tcl Structure，基于 *“参考2”* 重画

从图中我们可以了解到有个概念很重要，你输入到Tcl的所有交互的指令或者其他接口都被认为是文本，按一定的顺序进行Parsing。这个特性在某些应用场景变得很有用，命令也是数据（某些其他语言在命令和数据方面的互通性更好）。

EDA软件对Tcl的支持现在都已经到8.6了，由于其扩展性好，调试简单，C接口友好，使用者学习成本低，这些都是商用软件的价值所在，最重要的是开源，免费。我们看到在IC设计流程中，有些文件格式是基于Tcl的，比如SDC。

Date	Event
January 1990	Tcl announced beyond Berkeley (Winter USENIX).
June 1990	Expect announced (Summer USENIX).
January 1991	First announcement of Tk (Winter USENIX).
June 1993	First Tcl/Tk conference (Berkeley). [table] geometry manager (forerunner of [grid]), [incr Tcl], TclDP and Groupkit, announced there.
August 1997	Tcl 8.0 introduced a bytecode compiler.
April 1999	Tcl 8.1 introduces full Unicode support and advanced regular expressions.
August 1999	Tcl 8.2 introduces Tcl Extension Architecture (TEA)
August 2000	Tcl Core Team formed, moving Tcl to a more community-oriented development model.
September 2002	Ninth Tcl/Tk conference (Vancouver). Announcement of starkit packaging system. Tcl 8.4.0 released.
December 2007	Tcl 8.5 added new datatypes, a new extension repository, bignums, lambdas.
December 2012	Tcl 8.6 added built-in dynamic object system, TclOO, and stackless evaluation.

我们现在最新使用的也是Tcl8.6，最近一次发布时Jul 27, 2016。

3. 三十而立的Tcl 为何依然发光发热

看点：优点并不是任何场合都适用的，天天和IC、Tcl 打交道的你看看我对Tcl 优点的总结。

- 免费

免费，尤其是对商业免费是很重要的，企业对轮子的选择，如果不想自己造，那就要规避产品的法律风险，而免费开源（某些协议）首先消除了企业的这一顾虑，开源更是给了企业更大的自主权，可以利用整合的力量。免费开源这一点对于我们个人生活工作中对工具或者学习材料的选择，就理性而言（感性除外），需要考虑风险，成本与产出；任何人的时间和关系都是成本，企业更加关注成本这类关键字。

- C 的结合紧密

Tcl 是用C 语言开发的。它现在可运行在Unix，Windows 和Macintosh 等各种平台上。与C/C++ 的精密结合，互相调用方便进行切入。高可用的商业软件使用C++ 开发的很多，而选择一种脚本语言可以做为高级语言的辅助工具也很重要。从Tcl 中访问C 语言可以在外部为程序临时打补丁；相反，从C 语言中方位Tcl 可以将输入内容传递给内部功能。你中有我，我中有你的配合虽然在最终用户处使用场景不多。IC 设计需要调试（Debug）具体设计，还要调试（Debug）工具特性，这种异常复杂多变的场景，Tcl 为其提供了类似于在线调试的能力（在线意思为实时互相通信），充分利用了编译语言的可靠性和脚本语言的灵活性。

- 开发部署周期短

脚本语言是解释性语言，不需要编译，这个是脚本语言通性，这个特性保证了支持团队的反应时间。如果工程师在设计中遇到一个问题，发给支持团队，可能是EDA 公司，可能是公司内部的CAD 部门，他们需要重新编写一些功能，对代码进行编译，测试，然后再发给到出问题的工程师手里，如果还是不能解决问题，则需要继续迭代，这个沟通过程

当然有点典型和夸大的成分，非常没有效率；但如果使用脚本语言，不触及最核心的高级语言功能代码，只是对外部的东西进行处理，提供一个接口与内部进行沟通，脚本的测试和调试也要快很多。总结来说就是开发和部署的周期大大缩短，适合IC 行业，IC 行业中的核心功能算法不需要快速迭代，AE 或者CAD 工程师面对的是如何将客户的数据或者输入格式调整到工具能够识别的最佳状态，找到一个最好的输出Result。

- TK 图形界面

Tk (Tool Kit) 是基于 Tcl 的图形程序开发工具箱，是 Tcl 的重要扩展部分。Tk 隐含许多 C/C++ 程序员需要了解的程序设计细节，可快速地开发基于图形界面 Windows 的程序。据称，用 Tcl/Tk 开发一个简单的 GUI 应用程序只需几个小时，比用 C/C++ 要提高效率十倍。需要指明的是这里所说的“窗口”是指 Tcl 定义的窗口，与 X-Windows 与 MS Windows 的定义有所不同，但它可完美地运行在以上两个系统上。

TK 做为各种脚本语言基本上都支持的一种图形界面，它的Component binding 等思想到现在都是很好的方法。图形界面是和用户进行交流的一个重要工具，TK 最为一个快速的模型，可以很简陋也可以很优雅。在最后一个部分，我将会介绍几个TK 的著名程序，除了使用它们，阅读它们的源代码也会受益匪浅。

- 调试性语言

把Tcl 叫做调试性语言，是因为Tcl 做为一种粘合工具，很容易开发出一套访问核心数据的接口命令，比如EDA 工具的get以及*report*系列命令。获取信息后配合set_系列命令，快速调试环境，继而配置需要的内容。而Tcl 本身作为程序语言，虽然比不上Lisp 等语言完备，但是应用起来也不成问题，所以简单这个特性在自动化测试（Test），调试（debug）中便成了Tcl 的一个优势。

- 整合性语言

Tcl 和C 方便的接口，让它拥有了强大的整合能力。撇开C 语言，即使Tcl 做为单纯的外部Wrapper 工具（套壳工具）也是合格的，这里我指的是利用Tcl 将不同的可执行程序，或者不同语言的脚本整合到一起，Tcl 本身就是为这个而设计的，有一套处理异常的方法和完整的输入输出能力（当然这个并不是多大的优点）。我习惯给核心程序外加一层命令或者函数去判断其是否出错，（有些编译型的程序语言的错误处理当然必须先编译），对于line by line 执行的Tcl 来说，必须加一件外衣才能去catch 命令的错误。

- 入门简单，转移关注度

学习成本低对于商业软件来说价值很大，对IC 工程师尤为重要。一个复杂的语言，学习成本高，迭代周期长导致迭代成本高，不利于问题的分解，容易让工程师偏离IC 设计的主要矛盾 - 设计本身的PPA。这个特性也是Tcl 发明的最主要动力，很多工程师的Tcl 知识其实到第一个level 就够用了，我一开始的十几个个问题，能得到5-6分应该就算第一个level 了。

- 扩展性强（package 的优势）

Tcl 的Package 和其他程序语言的库都是很像的，而且Tcl 自身也提供了一套如何维护库的命令。Tcl 的库如果不涉及编译的话（比如需要访问数据库）是纯文本，可以说很简陋，当然你也可以把这个扩展用得很高大上。从简单的思想拼凑到优雅很直接，而简化一个复杂的结构是有一定难度的。

- 跨平台（Tcl 本身）

很多系统操作命令，不同平台的方式都是充分利用操作系统的资源，包括TK 这个界面程序也是一样，在保证功能一致性的情况下，无需编译便可以跨平台运行。

- 网络功能（Tcl 本身）

Tcl 通过http 包可以方便地进行网络访问，大家可能有用python 或者任何其他语言做爬虫的经验，Tcl 也是可以的，但是如果用在IC 上，用在系统的构建上，这个能力也是不可或缺的。

4. IC 工程师为何要熟悉Tcl 等脚本语言

看点:一点学习**Tcl** 的体会，即使你厌恶编程，为了提高效率，为了绩效，为了工资，为了面试，你应该看一眼。

前面我列出了十几个问题，涉及了Tcl 的不同方面，仅涉及对程序语言的熟悉，不涉及各种额外的编程思想。

我曾经对物理设计的职位做过词频统计，去除了一些语法词以后，Physical 当然出现最高，Tcl、Perl 的频率也排在前列。

1.3.1.3 - Programming Languages

- Tcl
- Lisp
  - Scheme
  - SKILL
- Perl
- Python
- Shell

这些Scripting Language 都有各自的背景，Tcl 已经说了很多了，其产生的原因就是为了解决和电路设计系统的交互问题。而像Perl，Python，用于处理报告和做数据挖掘、机器学习有它天生的优势。

Lisp 虽然复杂，虽然我也没用过，但是我了解它函数式的特性以及数据和命令的一致性很给力，很多模拟人都用SKILL或者SCHEME，Lisp 的子语言。

我们的工作就是和设计打交道，和工具打交道，脚本语言是设计系统，操作系统开放给我们的交流工具，我们还做不到要系统来理解人类语言，所以只能反过来去学习系统能懂的语言。

4.1. 如何学习

如果有编程基础，那么学习一门编程语言并不复杂。对于我们学习Tcl，我分为三个级别：

总的来说，挑一本Tcl 的书，从前往后看，你就能掌握Level 1 到Level 3，当然光看不行，还要动手实践。在你到Level 3 以后，相信你自己已经有很多Tcl 的脚本积累了(如何积累，以及管理这些积累也是一个比较大的话题)。

1.3.1.4 - 3 Levels

Level 1：和EDA 工具打交道

- 大概了解Tcl 是如何工作的，比如前面提到的，每个命令都是文本，都需要parsing。
- 会使用语言的基本命令，变量定义，条件语句，循环语句，异常处理等（一般书的前几章）。
- EDA 工具的命令需要去多看，多用（多翻工具手册）。

Level 2：实现一些复杂的功能

- 常用命令的基本用法和Tricky 的用法都要会。
- 会使用包，熟练使用文件处理和正则表达。
- 理解Tcl 运行的Level 和Domain。

Level 3：编写界面及复杂的应用

- 会用进程，以及程序间通信。
- 使用常用的包链接更多功能，如数据库。
- 制作TK 界面

每个级别并不是只有三个技能，限于篇幅我只是列举出比较典型的。

“有没有捷径可以走”，悄悄告诉你，“我有，请往下再看两到三章”。

5. Tcl 家族：Tcl/Tk/Expect

看点：只用**Tcl、Tk、Expect** 也可以创造一个自己的**IC** 黑客帝国。

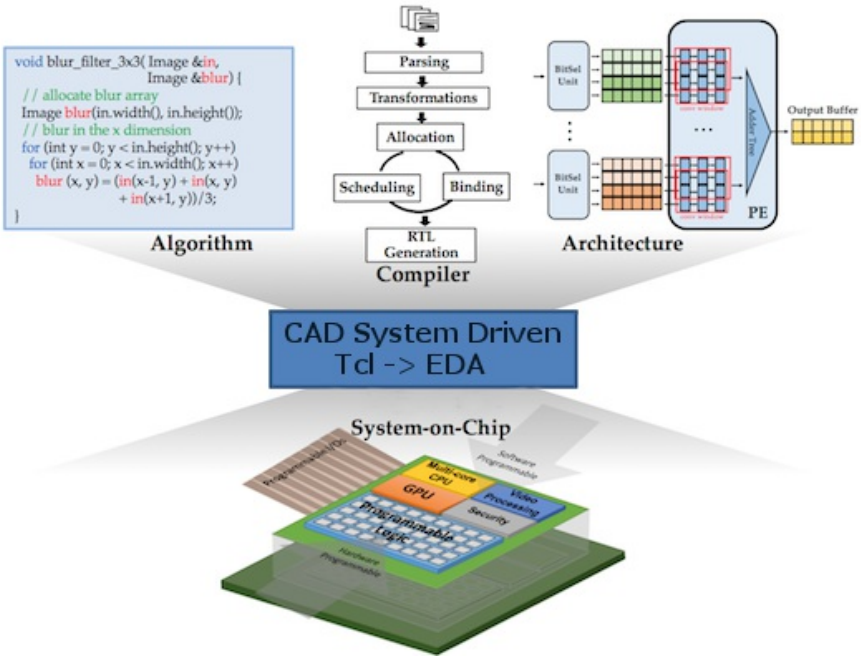
Expect 是Tcl 的一个扩展，或者说是Tcl 的另外一个可执行的分支，它的主要作用是用来粘合那些需要有互相交互的程序。

比如你登录某个项目需要输入密码；比如每次需要输入某个特定的项目名字，这些需要交互的内容可以用expect 来进行链接，普通的pipe 只解决输入输出的链接，对交互没有办法。

你是不是已经跃跃欲试了，从此可以解放你的很多冗余操作。

Tk有一个Bwidget 的Package，所有都是用标准的Tk 组件编写一个扩展，好处在于方便修改，方便升级和维护。

Tcl 用来实现基本的逻辑功能，应用；Expect 用来解决自动化中的交互；Tk 用来编写界面。你想造一个自己的系统用Tcl 系的程序语言就可以实现了。



1.3.1.5 - Tcl System

通过学习Tcl/TK/Expect：

1. 更好地与EDA 工具交互，扩展EDA 的命令集。
2. 可以扩展EDA 的界面，虽然没有QT 那么华丽，但是基本功能都健全。
3. 可以用Tcl 访问数据库（mysql，sqlite3，neo4j）。
4. 可以用Tcl 编写各种逻辑功能，完成基本的算法。
5. 更加自动化你的工作，提高效率。
6. 访问网络，和Web 协同，数据通信，应用衔接。
7. ...

6. 为什么Tcl 也可以做系统

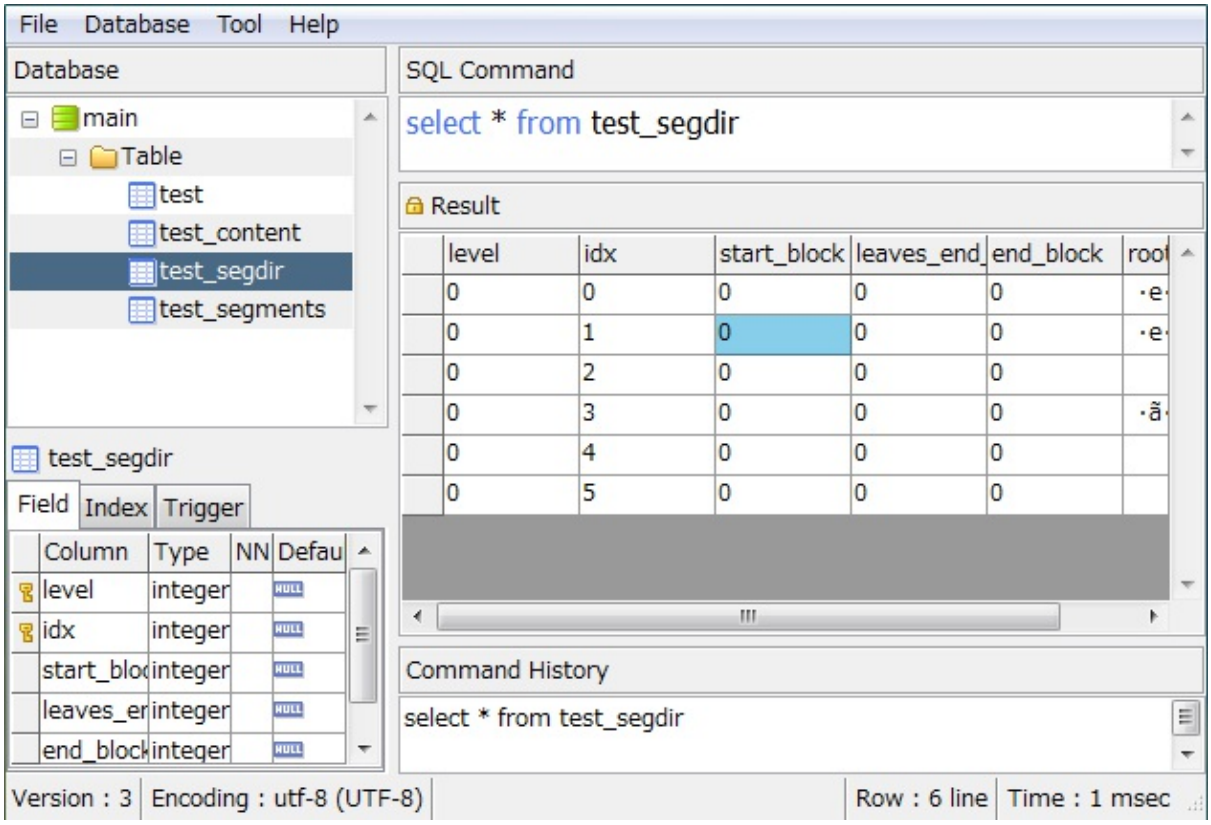
- EDA 原生支持Tcl，可以有很好地交互和集成。
- 做为系统，界面是一个很重要的交互。
- Tcl 有丰富的接口，而且很容易扩展，接口方面和数据库的接口很齐全。
- 方便扩展EDA 功能，形成一套自由的体系。
- Tcl 的粘合剂作用可以方便的将不同的可执行程序拼接起来。
- Tcl 命令和数据基本可以混用，方便数据和命令在同一程序中混合。
- 基于上面一点，命令可以是字符串，是数据也是命令，方便应对各种变数。
- 逻辑层和数据层分开。

7. Tcl 优势的几个实例推荐

有几个常用的TK 工具，很好用的工具，同时它的源代码也值得一看。由于本章节更偏重实践，所以微信中为节选，可查看原文获取完整内容。

7.1. tksqlite

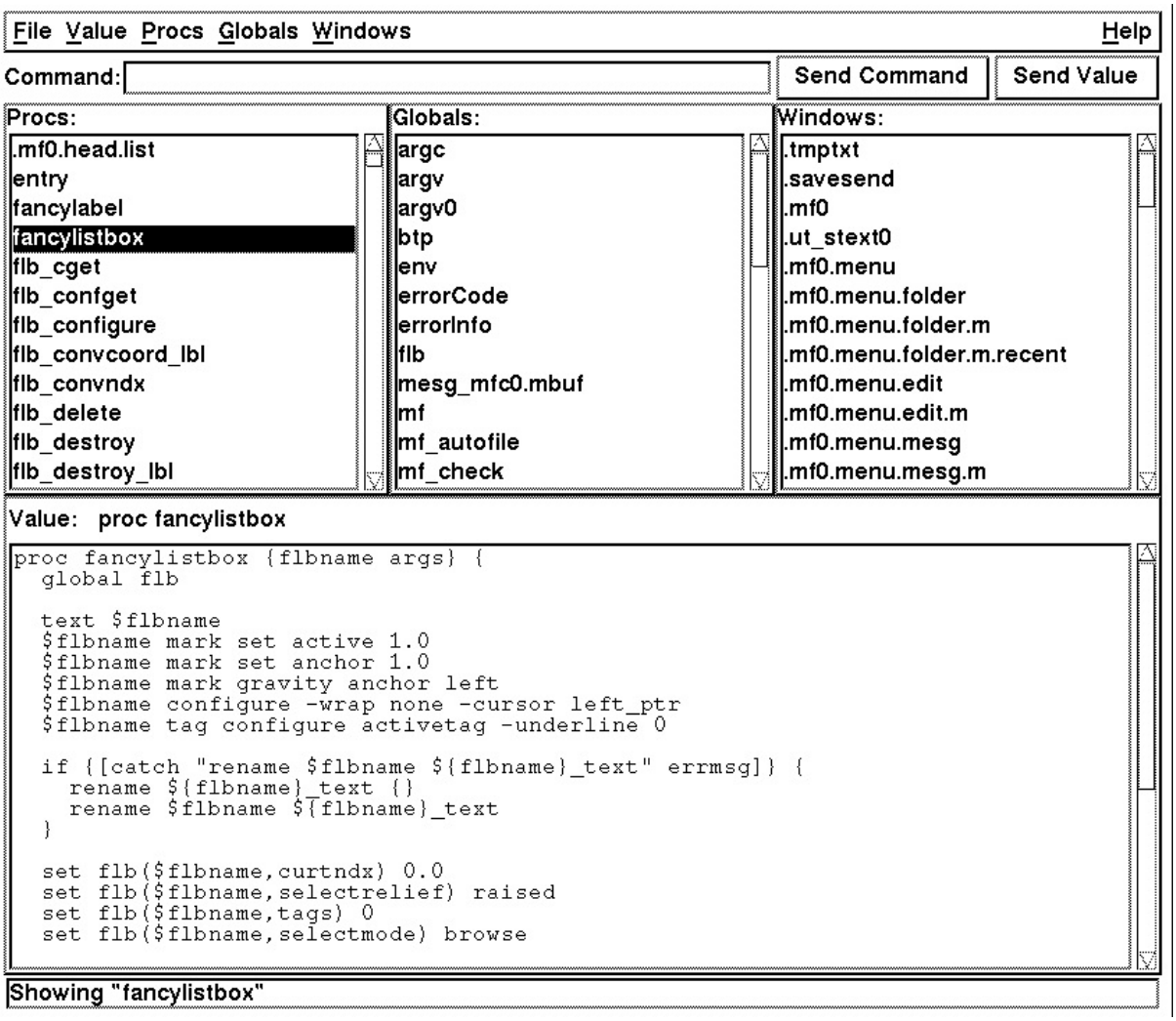
sqlite IDE，这个是对sqli数据编写的一个很好的工具，是学习数据库访问的一个的很好地实例。



1.3.1.6 - TKSqlite

7.2. tkinspect

TK inspect tool，这个程序可以分析自己编写的TK 的各个namespace，各个component，这个tool 是Tk应用交互的一个很好的实例。



1.3.1.7 - tkinspect

8. 文末思考题

Tcl 的好处我也鼓吹那么多了，那么如果你有一个私人Support 团队，CAD 团队为你服务，

你希望有一个什么最能解决你当前问题的Tcl 脚本或应用？大家可以留言讨论或者入群讨论。

这个假设是你有一个强大的Support 队伍，如果没有呢，你如何自己实现？

9. 参考内容

- SteveB 笔记
- 《Tcl & the Tk Toolkit》
- 《Tcl/Tk : a developer's guide》
- 《Practical Programming in Tcl and Tk》
- http://www.wikiwand.com/en/Scripting_language
- https://blog.csdn.net/larryliuqing/article/details/20902181
- http://scc.qibebt.cas.cn/docs/linux/script/TclTkall.pdf