Tcl/Tk as High-Level Control Language for Embedded Devices

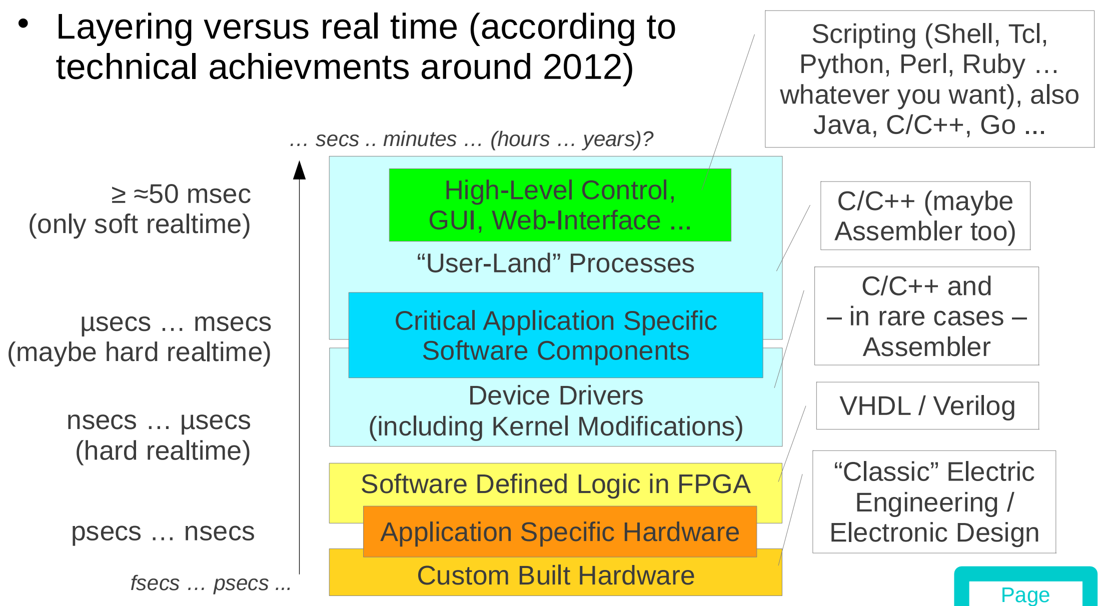
Dipl.-Ing. Martin Weitzel
Technische Beratung für EDV
64380 Roßdorf, Germany
`www.tbfe.de`

Architectural Options for Embedded Devices

- Programming the “Bare Metal”
- As before plus some chip-vendor libraries
- “Home-Grown” tiny OS
- Industry Standard OS
  - Windows CE
  - Linux / BSD-Unix
  - and other “RTOSes” (picked from German Wikipedia):
    - QNX, VxWorks, Nucleus, …
    - OSEK, OS-9, RTEMS, ECOS ...

Embedded Linux Design Architectural Layers

- Layering versus real time (according to technical achievments around 2012)



Scripting (Shell, Tcl, Python, Perl, Ruby … whatever you want), also Java, C/C++, Go ...
… secs .. minutes … (hours … years)?
≥ ≈50 msec (only soft realtime)
High-Level Control, GUI, Web-Interface ...
"User-Land" Processes
C/C++ (maybe Assembler too)
µsecs … msecs (maybe hard realtime)
Critical Application Specific Software Components
C/C++ and – in rare cases – Assembler
Device Drivers (including Kernel Modifications)
nsecs … µsecs (hard realtime)
VHDL / Verilog
Software Defined Logic in FPGA
"Classic" Electric Engineering / Electronic Design
psecs … nsecs
Application Specific Hardware
fsecs … psecs ...
Custom Built Hardware

Why Use a Scripting Language?

- No compilation an linking
  - Fast turn-around in development cycle
  - Lesser hassles with the tool-chain
- Different learning curve
  - initial steps typically less steep ("learning while doing")
- Option for *Rapid Prototyping*
  - All modern scripting languages can be easily extended with C/C++ modules (cf. http://www.swig.org)
  - **No impasse if performance turns out to be insufficient!**

Why NOT Use a Scripting Language?

- Typically much less type checking – or no type checking at all
  - Design errors may manifest themselves not before run-time
  - The condition(s) triggering the error may depend on customer data and hence have slipped through testing
- Scripting may be considered “amateurish” by customer
- Good Advice:
  - **Invest some of the time saved in the development cycle into extended testing**
  - Write test suites than can be run automatically and add new tests against every error you find past alpha-release

Why use Scripting for High-Level Control?

- High-Level control is often least time critical (no hard real time)
- If customer dependant changes affect that level ...
  - … new Customers may be quickly presented a prototype
  - … scripting lends itself naturally to *Agile Development*, where proposed changes are prioritized by usefulness, i.e.
    - discuss an idea or requirement with the customer ...
    - … follow it with a “quick and clean” implementation for closer evaluation, and only if considered useful ...
    - … make it robust and maybe add the “bells and whistles” which your customers might like

Why use a Scripting Language for …

- Graphical User Interface (GUI) ?
  - Typically no performance loss as most the “time critical stuff” is written in a compiled language anyway
  - Especially in Tcl: small ideas might be tried out interactively
- Networking
  - Socket programming in C/C++ is known to be hard
  - Most scripting languages make it more convenient
- Web-Interface (see above and ...)
  - Scripting languages somtimes offer “ready-to-use” tiny, small, or even full-blown HTTP-servers as add-on

Why use Tcl/Tk?

- Maturity
  - Clearly, Tcl has not much of the latest “trendy hype” …
  - … but being around for about 20 years it has proven its stability and reliability and is surely free of teething trouble
- Especially shallow first step of training curve
  - Tcl's “syntax” is minimal, most of it is “command” ...
  - ... i.e. library functions with knowledge acquired as needed
- **There is a double pay-off for FPGA developers:**
  - **Attained knowledge is applicable to many of their tools!**

Why NOT use Tcl/Tk?

- Tcl is an ageing tool
  - The downside of maturity is Tcl draws not as much attention as it's more recent cousins like *Python*, *Ruby*, *Go* … etc.
  - It might become increasingly difficult to hire experienced and enthusiastic Tcl developers
- Tcl's minimal syntax (You better not try to convince heretics ☺)
  - Unusual at best and maybe “archaic” to dedicated followers of more recent programming languages
  - **To make a Tcl application really robust it should receive a bit more test coverage compared to one written in a syntactical more strict language**

Comparing Tcl/Tk to the Linux-Shell

- There are many Pro's …:
  - Much more regular – not to say: really elegant – syntax
  - Much richer choice of data structures
  - “Batteries included” (Networking, GUI, … to name just two)
  - Easily extensible with C/C++ modules
  - With *TclX* access to most Unix/Linux system calls
- … and hardly any Con's:
  - Tcl is a separate package to install while a shell is typically always present (but maybe stripped-down as in “Busy-Box”)

Comparing Tcl/Tk to Python

- Python has a similar history as Tcl
  - Grown in a niche with little attention …
  - … until matured enough to go “into the wild” on its own
- Python also has a slightly “unusual” syntax but a stricter one as Tcl (so syntax aficionados will usually know where to go … ☺)
- In the meantime, Python surely has more followers as Tcl/Tk ...
  - … though one of Pythons prevalent GUI libraries is *tkinter ...*
  - … which is nothing else but Tk wrapped into Python syntax

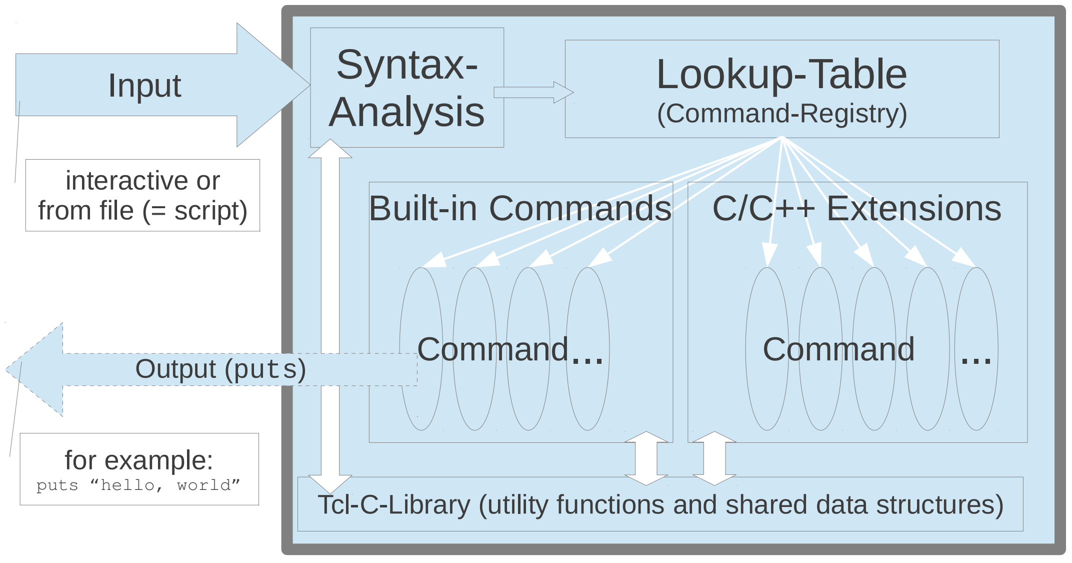
Comparing Tcl/Tk to …

- **Perl:**
  Without any doubt, there is a large amount of Perl-based software but as a language it looks even more aged as Tcl
- **Ruby:**
  As a late delivery it still enjoys the advantage not to have to care as much for backward compatibility as its competitors
- **Java:**
  No interpreted languages and actually a heavy ones now
- And well, there are many other languages looking promising or at least interesting, like *Lisp*, *Lua*, *Haskell, Scala, Go* …

A Whirl-Wind Tour Through Tcl

- Tcl's Syntax is minimal
  - Instead, for most anything there is a command
  - This includes arithmetic evaluations and flow control
- As an interpreter Tcl offers access to all of its internal state
  - Therefore tracing and debugging hooks are natura
  - Also sand-boxing for “unreliable guest scripts” is possible
- Tcls lends itselfs easily to extensions, including any mix of
  - extensions in “pure tcl” that simply get `source`-d
  - extensions in C/C++ that get `load`-ed as shared library

Whirl-Wind Tour: Internal Structure


Input
interactive or from file (= script)
Output (puts)
for example: puts “hello, world”
Syntax-Analysis
Lookup-Table (Command-Registry)
Built-in Commands
C/C++ Extensions
Command...
Command ...
Tcl-C-Library (utility functions and shared data structures)

Whirl-Wind Tour: Syntax Analysis

1. Join continued lines
2. Find end of command line
3. Split command line into words
4. While observing a tiny set of quoting rules substitute
   - unprintable characters (like `\n`, `\t` ...)
   - content of variables for `$`*`varname`* and
   - return values from called functions, determined by parsing commands in `[ … ]` with recursive syntax analysis
5. Execute command determined by first word in command line

Whirl-Wind Tour: Data Structures

- *Plain* variables, dynamically typed
  - strings (any content, even “`\0`”-bytes in more recent Tcl)
  - integral (with at least 32 bits signed)
  - “long” integral (unlimited precision in recent Tcl)
  - floating point (typically IEEE-754 64-bit representation)
- `array`-s (associative index, so rather “hashes”)
- `list`-s (semantically closer to C/C++ built-in arrays)
- `dicts`-s (modelling the *composite* design pattern)

Whirl-Wind Tour: Flow-Control

- Flow-Control supports the classic constructs
  - Branching with `if` – `else`, including `elseif` chaining
  - Multi-way branching (`switch`)
  - Repetition with `while` and `for` (close to C style)
  - Collection (list) processing with `foreach`
- Fast escape from errors and recovery as required
  - Modelled similar to (C++/Java/...) exceptions …
  - … though less “sophisticated”
  - … but much easier to handle

Whirl-Wind Tour: Subroutines

- Subroutines parameters include variable length argument lists
  - Default is *call by value*
  - Optionally caller's variables may be accessed and modified (*call by reference*)
  - Keyword arguments may be modelled in Tk-style
- Subroutine return values
  - Technically limited to plain variables …
  - … but not a real limitation since every structured type can be easily turned into and converted from a string

Standard Library

- String processing including regular expressions
- File processing modelled close to C/Posix style
- Networking (makes using TCP/IP sockets a snap)
- Supports event-driven design style (favoured – no threads!)
- Rich introspection / reflection features
- Slave interpreters for sand-boxing
- Sophisticated virtual file system (kind of a “hidden jewel”)
  - probably not exploited in many Tcl applications where it would be helpful or could provide very elegant solutions

Unix/Linux Specific Extensions

- Distributed applications may easily communicate with `send`
  - Restricted to Unix/Linux with GUI running because …
  - … built on top of X11's event distribution mechanism
  - Portable alternative: TCP/IP-sockets or `::comm` (Tcl ≥ 8.3)
- Extended Tcl (optional extension)
  - Gives more or less direct access to many system calls ...
  - … at the price of sacrificing portability
  - Formerly a separate interpreter (`tclx` instead of `tclsh`) …
  - … now a loadable shared library

Tcl Networking in General

- Based on TCP/IP-sockets
  - Very easy to handle:
    simplistic C/S-Application implemented in a hand-full LOC
  - Perfect match with event-driven designs favoured by Tcl

```
set s [socket 127.0.0.1 4712]
puts [gets $s]
close $s
```

“Greet”-Client and ...

```
proc p {sock adr port} {
    puts $sock "hello!"
    close $sock
}
socket -server p 4712
vwait forever
```

… “Hello”-Server

Web-Interface with Tcl

- Simplistic Web-Server implemented from scratch in 30 minutes
  - Code printed out fits on a DIN-A4 paper sheet …
  - … in a 10 or 12 pt. font …
  - … with still plenty of space for hand-written notes
- Various free (open source) solutions
  - Offering different levels of sophistication
  - Most advanced: AOLserver (guess customer from name!)
- Also browser plug-in for client-side Tcl programming (“tclets”) is available (http://wiki.tcl.tk/12718)

Tk based GUIs

- Tk was the first (scripting) extension that made Unix GUI based on the X11 window system easy
  - GUI elements abstracted as widgets
  - All basic interaction elements supported ...
  - … though maybe not the latest-and-greatest fancy stuff
  - Layout managed by strategies, not pixel-wise
- Some widgets are very powerful and easy to use, e.g.
  - `text` actually not limited to sophisticated text representation
  - `canvas` all of the "mechanism" required for 2D graphics

Object Oriented Tcl

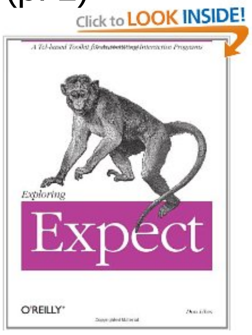
- Available in several flavours
  - *Incremented Tcl* (aka *[incr tcl]*)
    - Basically much like "C++/Java in Tcl syntax", therefore very(!) fast learning path with OOP foreknowledge
  - *Snit* ("A Truly Tcl Type System")
    - http://www.wjduquette.com/snit/ – last update 2005(??)
  - *XOTcl (Extended Object Tcl*, based on *OTcl*)
    - http://media.wu-wien.ac.at/ – last update 2011
  - based on *Otcl* (*MIT Object Tcl*)
    - http://otcl-tclcl.sourceforge.net/otcl/

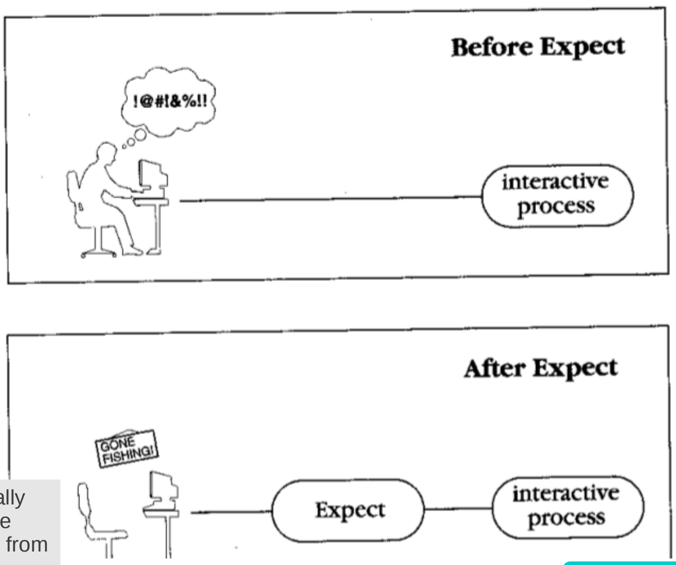
Oldest Tcl-Extension: Expect

- Written by Don Libes
  - http://www.nist.gov/el/msid/expect.cfm
  - Helps to automate any command line driven applications …
  - … i.e. not at all limited to applications written in Tcl
- Basic use: **`spawn`** an application and in a switch-like syntax ...
  - … describe what you `expect` as output and
  - … what you would `send` back in each case
- Portable across Unix/Linux, Windows, and Mac
- **Maybe another big – and "un-expect-ed" ☺ – boon if You choose to learn Tcl any purpose.**

What You Can Expect from expect

- Illustration from Don Libes' Book (p. 2)





The page with this illustration is actually excluded from the online version – the author of this talk shamelessly took it from his own copy of the book …

And now let's head for *“The golden tree of life ...”*

Questions and Suggestions are welcome at <u>any</u> time!

Grau, teurer Freund, ist alle Theorie
und grün des Lebens goldner Baum
© All rights reserved

That‘s All

Any (more) Questions?

Thank You for Participating