Comit's CVXT Tool

Venkat Iyer

Copyright Comit Systems, Inc.

Agenda

- Who am I
- What is CVXT
- What is Hardware Simulation
- How does Tcl/CVXT Help
- The Challenges and Solutions
- How Coroutines Helped
- Coro'ized CVXT Implementation
- CVXT Usage Example
- Conclusions
- Acknowledgements

Ask questions anytime. I stop when I run out of time.

Copyright Comit Systems, Inc.

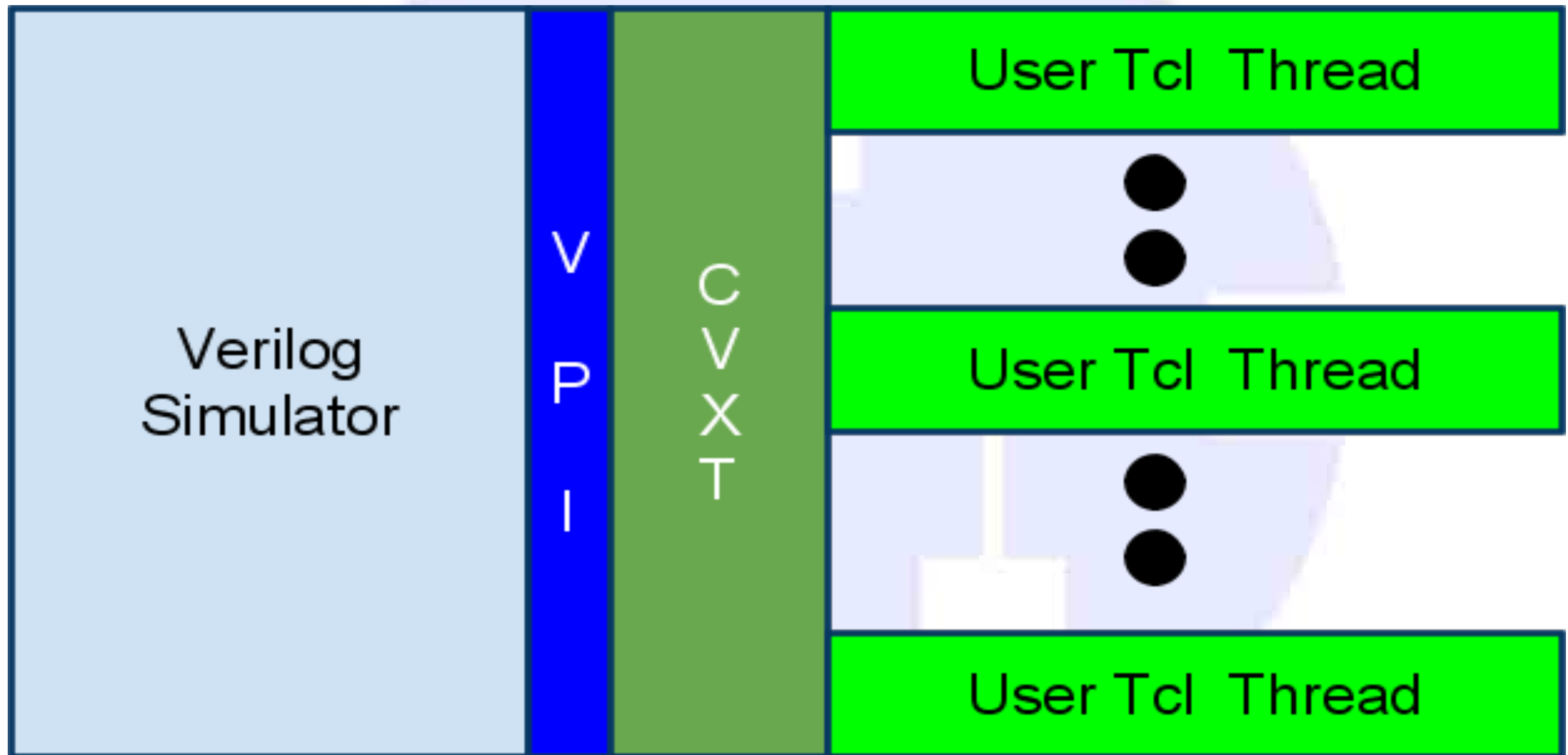
About the Author

- EE-like Undergrad, Comp Science Grad
- Main work-like Interests:
  - Compilers and Languages
  - Tools and Automation
  - Hardware Logic Design and Generation
- Tcl since early 90s 16-bit DOS Turbo C. Big Fan.
- http://wiki.tcl.tk/vi   - venkat@comit.com
- 17 years at Comit Systems, Inc: Chips/Boards/Systems/SW
  - Niche Contract Engg Firm in Silicon Valley.
- Comit uses Tcl for: Web Site, Internal Systems, EDA Tool-lets, Backup, ... everything

Copyright Comit Systems, Inc.

CVXT

- Verification Engine from the 90s
- Enables writing Tcl tests for hardware designs



Copyright Comit Systems, Inc.

Hardware Modeling

1. wait for a or b to change
2. compute a AND b
3. store the value in y
4. go back to step 1

This is called one hardware **process**

```
always @(a or b)
   y = a & b;
```

Copyright Comit Systems, Inc.

Hardware Simulation

- Event Driven Scheduling Kernels
- Millions of Virtually Parallel "Processes"
- Each Process suspended on signalling events or time
- Very good at modeling hardware.

Issues:

- Not designed for verification (many new efforts ongoing)
- Level of abstractness somewhere between C and assembly
- Testing extensions need more licenses ($$$$$$$$)
- New language requires new thinking

Copyright Comit Systems, Inc.

Why Tcl

- One less language. Most hardware designers know Tcl.
    - Most EDA tools use Tcl as the scripting language
- Portable, Built to be embedded
    - Easy to support multiple simulators and platforms
        - vsim, ncsim, vcs, cver, icarus.
        - 32/64. win/lnx/amd64/sparc
- Event driven (more later)
- Dynamic
    - Saves costly HDL re-compile times if tests change
- Easier OS services access
    - display images as they're processed
    - send sniffed packets into the simulator
- tcltest

Simple Example

"Thread" runs one user context, typically runs tcltest on a part of a design.   Say an ethernet interface.

```
set r [get tb.ethernet0.error]
if {$r == 0} {
   put tb.ethernet0.txen -value 1
   set w [wait -signal tb.ethernet0.eof -time 1000]
   set tb.ethernet0.txen -value 0
   if {$w eq "time"} {
      error "Packet was not transmitted in 1000 ns"
   }
}
```

Copyright Comit Systems, Inc.

Challenges

Mainly due to supporting various simulators from different vendors on multiple platforms.

- Threads
  - pthread libc incompatibilities
  - ucontext with thread-enabled Tcl, windows.
- Multiple Tcl Versions in one Process Space
  - Still support 8.3.4 in the simulator
  - CVXT runs bleeding edge.

Copyright Comit Systems, Inc.

Multiple Tcl Versions in One Process

- Build Tcl enabling shared support.

```
$ ./configure --enable-threads –enable-shared
$ make
```

- Link forcing resolution

```
$ gcc -m64 -Wl,-Bsymbolic -o cvxt.so <cvxt objects> \
  <tcl core objects> -shared <platform specific -l flags>
```

- cvxt.so is loaded into the simulator (from command line)
- Initialization is a script-mod Tcl_AppInit + startup script
- Extensions to cvxt are built with -DUSE_TCL_STUBS

Enter Coroutines

The perfect match

- no more threads. --disable-threads worked
- simplified build (no ucontext emulation on windows)
- faster threads (about 10% improvement over 10000 switches)
- enables multiple contexts per thread
  - called branches, which share globals/procs/..

Following slides explain CVXT Implementation using coroutines.

Coro'd CVXT: Thread Creation

```
proc create_thread args {
  set u0 [interp create u0]
  foreach cmd [list get put] {
   $u0 alias $cmd $cmd
  }

  $u0 eval {
    proc wait args {
      return [yield [concat u0 $args]]
    }

    proc start {} {
      catch {
       user code here sourced from $args.
      }
    }
  }
  process [$u0 eval coroutine __run__ start]
}
```

Copyright Comit Systems, Inc.

Coro'd CVXT: Thread Switching

```
proc process l {
  set child [lindex $l 0]
  foreach arg [lrange $l 1 end] {
    add call backs into simulator,
    and remember in data structure
  }
}

proc callback args {
  # called by simulator
  figure out which threads need to be awakened
  foreach interp $wakeup_threads {
     process [$thread ::__run__ $args] ;# resume the coroutine.
  }
}
```

Copyright Comit Systems, Inc.

CVXT: TclOO Example, Definition

```
class create mailbox {
   constructor {} {
      set name [namespace tail [self object]]
      if {[catch {set mbdata $shvar(mailbox_$name)}]} {
        set mbdata [list]
      }
      set shvar(mailbox_$name) $mbdata
   }
   method put args {
      lappend shvar(mailbox_$name) $args
   }
   method get {} {
      while {![llength [set mbdata $shvar(mailbox_$name)]]} {
        wait -shvar mailbox_$name
      }
      set shvar(mailbox_$name) [lrange $mbdata 1 end]
      return [lrange $mbdata 0]
   }
}
```

Copyright Comit Systems, Inc.

CVXT: TclOO Example, Usage

Server

```
mailbox create  reg_writer
while 1 {
  lassign [reg_writer get] mbox op addr data
  do the register transaction in simulator or hardware
  mailbox create $mbox
  $mbox put $read_data
}
```

Client

```
mailbox create  reg_writer
mailbox create  reg_response
reg_writer put reg_response read 0x30943544 0x33
set reg_value [reg_response get]
```

Copyright Comit Systems, Inc.

Conclusions

- Coroutines are good
  - They make multicontext event driven programming simpler (and in my case faster)
- It's good to piggyback on a language like Tcl
  - Has well thought out features that CVXT
- Tcl makes a good verification language

Acknowledgements

- John Osterhout for the Tcl Language
- Activestate, the TCT, maintainers and supporters of the Tcl Language and the Wiki
- Miguel Sofer for NRE.
- Many Tclers in the Chatroom
- Organizers of Tcl2009.