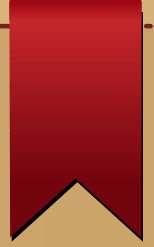
Gated-Trunk Hardware Development and DevOps

How we reduced ASIC development time by gating commits and including IT

Matt Dew aka marcoz
marcoz@osource.org

Overview

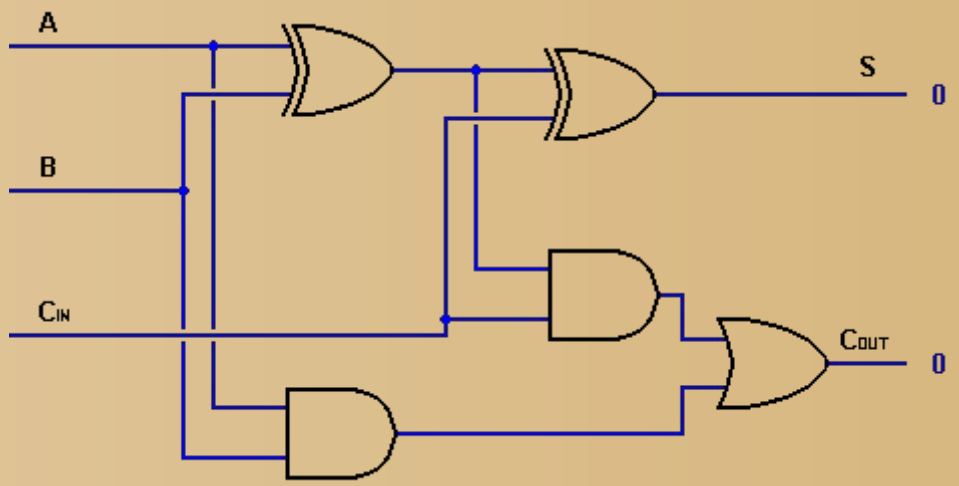


- Situation
- Gated Trunk
  - What is it?
  - Why do we use it in hardware development?
  - How do we use it?
- DevOps
  - What is it?
  - Continuous Delivery
  - Continuous Deployment
  - Treating IT like one of the gang
  - Gains
- How does this apply to X.org development?

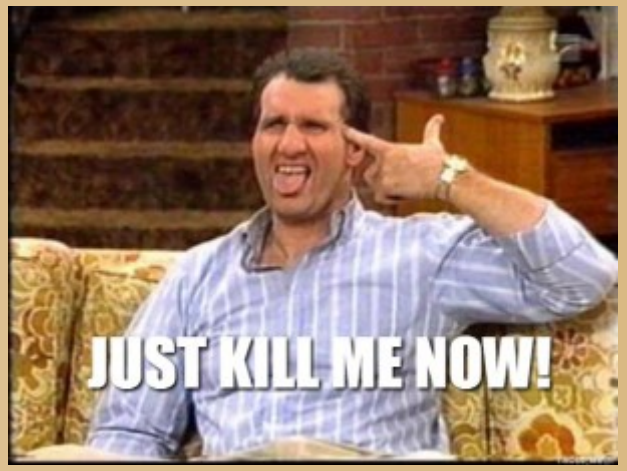
- Disclaimer - I have permission to speak but I do not speak for my employer. I am not representing my employer nor speaking in any official capacity. I am here as a private individual.

- I am part of the ASIC Verification group. I do design verification.
- I also manage the build flow and build environment

- Does anyone still think hardware developers still sit in their cubes and draw things like this?



Considering ASICs can be 150 million+ gates ...



System Verilog looks like:

```
module   full_adder_4bit(
    cin,
    cout,
    in_a,
    in_b,
    sum
    );

    parameter   reg_size = 4;

    input   cin;
    input   [reg_size-1:0] in_a;
    input   [reg_size-1:0] in_b;
    output  [reg_size-1:0] sum;
    output  cout;

    assign   {cout,sum} = in_a + in_b + cin;

endmodule
```

Situation

- > 150 engineers
- Geographically diverse
- Multiple areas:
  - RTL (designers)
  - DV (testers)
  - FPGA (validators)
  - ASIC (synthesizers)
  - ...
- Large designs consist of smaller components with different teams working on them.
- Need code reuse and development stability.
- **Stability and development speed; more people and more code = things slow down, things break more often**

- What have we used to help manage this? Several things, 2 of them are: gated trunk and DevOps.

Gated Trunk

aka read-only master branch,
aka Pre-emptive C.I.
aka Project Gating

Gated Trunk – What is it?

- What is it ?
  - Developers cannot commit directly to the mainline development branch. (svn: /trunk)
  - Instead work is done on feature/dev branches
  - To merge the branch into trunk, a merge is submitted to an automated gatekeeper for testing and merge.
  - Branches are (supposed to be) shortlived (keep C.I. benefits)
  - Submit raw patches and let it handle everything else?

Gated Trunk – Why?

- Why?
  - We still want “Commit early, commit often, fail local as soon as possible in a visible and obnoxious way” but we also want continuous delivery on a stable trunk; a codebase that anyone can checkout at any time and be able to use.
  - Code reuse - Multiple projects, sub-projects, functional blocks, etc; breaking one trunk has ripple effect to other things.
  - The same problems that bite many software groups also bite us, a hardware group.
  - We started with just continuous integration but breakage increased as the group size and codebase size increased.
  - Things like:

Gated Trunk – Why?

- Initially we had a system where people would run a pre-commit check suite before committing. It helped...for a while.
- Seemingly unrelated things would break each other.
- “This won't affect anything.“
- “My manager is breathing down my neck about this fix.”
- “Fred's commit just beat mine out and broke my stuff.”
- “It takes too long, I'll fix it if the C.I. runs fail.”
- “There are no licenses available, I'll fix it if the C.I. runs fail.”
- “Simulation ran, there are no synthesis checker licenses available. I'll fix it if the C.I. run fails”
- Sometimes a commit broke trunk and before it could be fixed, someone committed additional changes that depended on that first commit.

This led to...

Gated Trunk – Why?

Trunk manglers support group.

http://www.westword.com/location/tasty-weasel-tap-room-5173955

“Hi, my name is Matt and I've mangled trunk (again)”

Gated Trunk – How?

- How do we use it?
  - Jenkins has commit access to trunk, engineers do not.
  - Engineers work on branches, and use a Jenkins job to submit their branch for merging. The Jenkins job shows branches in a dropdown list box.
  - Jenkins runs a script which basically runs normal commands that the engineers can run.*
  - The engineers maintain which tests are used in the gate.
  - Change tests depending where in the project cycle we are. The list is kept with the codebase and evolves with the project.

* There is a Jenkins plugin that basically does this as well. We haven't moved to it (yet) because this process was in place before that plugin existed and we lack manpower to look into it vs something like Zuul.

Gated Trunk – Types of tests

- Several types of tests:
  - Simulations (to verify the RTL functionality)
  - FPGA synthesis checks
  - ASIC synthesis checks
  - Modeling checks
  - Firmware checks
  - Coding style checks
  - System resource limit checks (IE: If you need 128 GB of memory or 3 hrs to test read a register, you're doing something wrong.)
  - Dependency checks. (If you require certain jobs to run before others, you might have inefficient, or bad, dependencies in your jobs.)
  - Etc

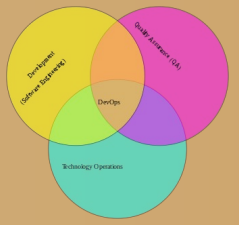
Gated Trunk – with C.I.

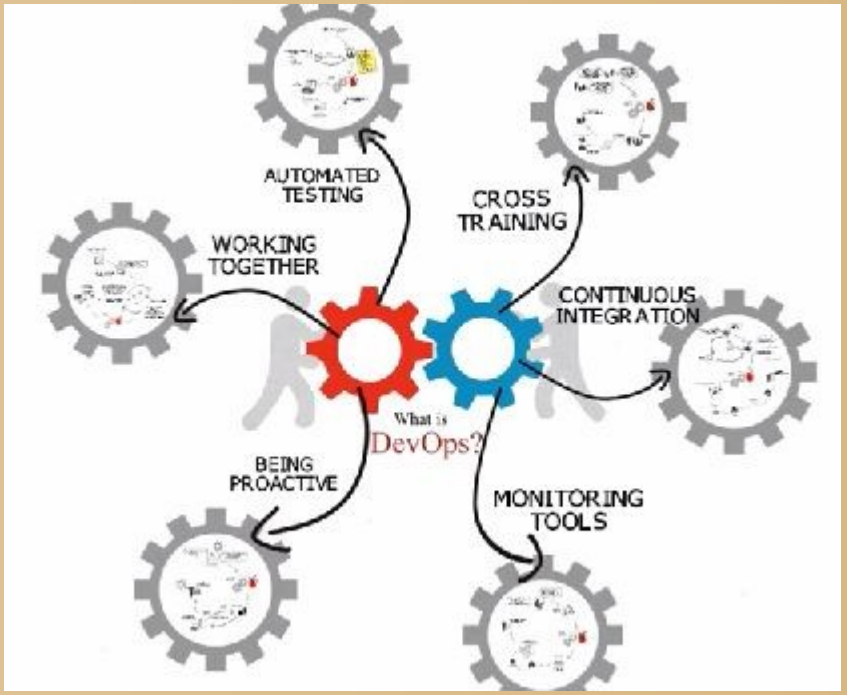
- With C.I. and nightly/weekly regressions.
  - Svn hooks still queue Continuous Integration jobs.
  - In addition to time, a finite number of licenses also forces a tradeoff of how often we can run certain tools on the change and/or codebase; gate vs C.I. vs regression

Not a silver bullet

- Gated Trunk helps us keep our trunk stable. Went from daily breaks to rarely breaks. Commit times slowed down, but development speed increased. (We spent less time fixing things.)
- But it doesn't solve all our problems.

DevOps




AUTOMATED TESTING
CROSS TRAINING
WORKING TOGETHER
CONTINUOUS INTEGRATION
What is DevOps?
BEING PROACTIVE
MONITORING TOOLS

DevOps

- What is DevOps?
  - 2008 Agile Conference – DevOps coined “Development” and “Operations”
  - From: https://en.wikipedia.org/wiki/DevOps
  - DevOps is a software development method that emphasizes communication, collaboration (information sharing and web service usage), integration, automation, and measurement of cooperation between software developers and other IT professionals.[1][2] The method acknowledges the interdependence of software development, quality assurance (QA), and IT operations, and aims:

    1. **to help an organization rapidly produce software products and services**
    2. to improve operations performance.

    tl;dr; Developers, QA and IT work **together** to get higher quality work done faster.

  - “DevOps is also characterized by operations staff making use many of the same techniques as developers for their systems work. “

- These slides are about how we use #1 in hardware development. (We end up getting some of #2 in the process.)

Continuous Delivery

- Continuous Delivery - “Continuous Delivery (CD) is a software engineering approach in which teams keep producing valuable software in short cycles and ensure that the software can be reliably released at any time.”
  - https://en.wikipedia.org/wiki/Continuous_delivery

Continuous Deployment

“Continuous Delivery doesn't mean every change is deployed to production ASAP. It means every change is proven to be deployable at any time

— Carl Caum (@ccaum) August 28, 2013”

“I think many people confuse "Delivery" with "Deployment".

— Magnus Hedemark (@Magnus919) August 29, 2013

- “Continuous deployment is the next step of continuous delivery: Every change that passes the automated tests is deployed to production automatically.”

  https://puppetlabs.com/blog/continuous-delivery-vs-continuous-deployment-whats-diff

Environment and Product
in hardware development

- Continuous Delivery applies to both the development environment and the product (ASIC) itself.
- Continuous Deployment is mainly applicable to the development environment.

Continuous Delivery
in hardware development

- Continuous Delivery in the environment is things like ensuring your environment modules are usable.
- Continuous Delivery in hardware development is much more than “does my simulation pass”. It additionally means the code must pass:
  - synthesis checks
  - FPGA validation checks
  - firmware checks
  - coding style checks
  - etc.
- Working with DV (test), RTL (designers), FPGA (validation), ASIC (synthesis), modeling, firmware, and other groups is a requirement to be truly continuously deliverable.

Continuous Deployment
in hardware development

- Continuous Deployment in the environment is things like:
  - Rolling out environment updates.
    - An update is committed, an svn hook queues up a jenkins jobs that sets up a fresh env for testing, tests it, on success, deploys the update to all the data centers.
  - Vendor IP updates
    - Compile code into libraries and deploy for engineers to link against. Reduce recompiles by pre-compiling (semi)-static libraries.
  - Using tools like puppet, salt, chef, ansible,

Pulling IT into the party

- DevOps means working with IT to achieve better development speed.
- to help an organization rapidly produce software products and services

  Takes down the wall and brings IS into the conversation alongside design, test, validation, firmware, etc.

  Agile + Data Center (System and Environment) Profiling + Flow Profiling + automation

- **to improve operations performance.**

  Helps IT but it allows that silo wall between ops and dev to remain. It just gives you a tauter tennis racket so you can throw things back and forth over the wall quicker.

  Agile + Automated Delivery and Deployment of IS

IT is an important part of the team

- “Bigger NAS/SAN and faster CPUs” is well meaning but we discovered it doesn't always improve things.

  We can saturate 20Gb network links and local eSSD bandwidth. Code and flow changes are sometimes more beneficial. However they are labor intensive, so system monitoring information provides data for discussions and prioritizations.

- LSF changes based on engineering flow.

  LSF per-user job limits prevented a single person from using all licenses but more often just left licenses unused and just slowed things down. A few engineers started depending on that limit; when we removed it, their jobs started failing in spectacular ways that were time-consuming to track down.

- System Verilog package imports are expensive (compile, optimization, elaboration).
  - System Profiling (correlating server monitoring [CPU, disk activity, network activity, …]) and LSF job information pointed us in the right direction.
  - We put effort into removing unneeded imports and into refactoring code to reduce existing ones.

- Network storage is easy for IT to manage. Local disks have lower latency; are better for interactive development.
  - Metrics are gathered to provide data for discussions about engineering wants and needs vs IT wants and needs.
  -
- Talk with engineers about cleaning out their areas. “If it's not checked in, it doesn't exist.”
  - Backing up everything encourages apathy and wasted disk space. The more that's backedup the more tapes are needed and the longer retrieval is.

- “That's seems counter-intuitive”, “That doesn't seem like it would work.”

  That's why we measure things and we consider IT as one of the groups in the team, not as a separate service delivery organization.

Development Gains

- Svn commit gains
  - before feedback - 15 minutes
  - after feedback - 100 milliseconds
- System Verilog Package Optimization gains
  - Before feedback – 12 seconds
  - After feedback – 0.5 seconds
- Average NAS usage
  - Before feedback > 14TB
  - After feedback < 1 TB

Summary

Biggest overall benefit is the reinforcement that each group is a piece of the larger puzzle. If we:

- worked together rather than throwing things over the wall to each other
- Make our stuff available to each other as soon as possible
- keep each other's needs in mind

then we get things done faster with less frustration and better results.

How does this apply to X.Org?

- It benefited my group. I thought it might help others.

Additional Resources

https://en.wikipedia.org/wiki/DevOps

http://www.arrkgroup.com/thought-leadership/what-is-devops/

https://www.kavistechnology.com/blog/wp-content/uploads/2013/08/devops2.png&f=1