Development of a PCIe DMA engine verification framework

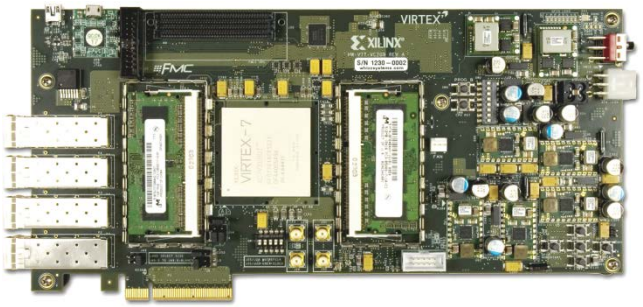
Michal HUSEJKO
eda.support@cern.ch

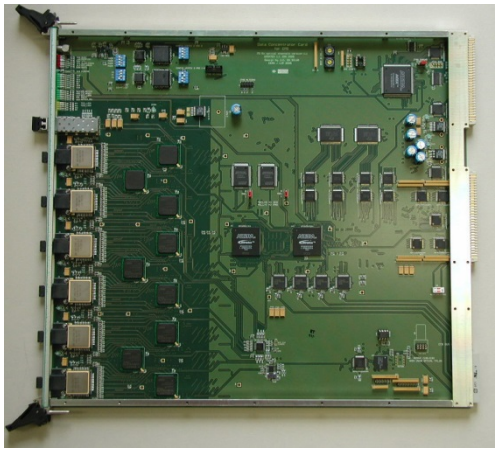
Introduction

- About me:
  - Working in the CERN eda.support
  - I’m responsible for “Introduction to VHDL” course available from CERN Technical Training Catalog
  - I’m also involved in different IT projects
- Trying to stay up to date with (or learn):
  - VHDL, tools
  - SV+UVM
  - Software development and Contiguous Integration

Typical data concentrator

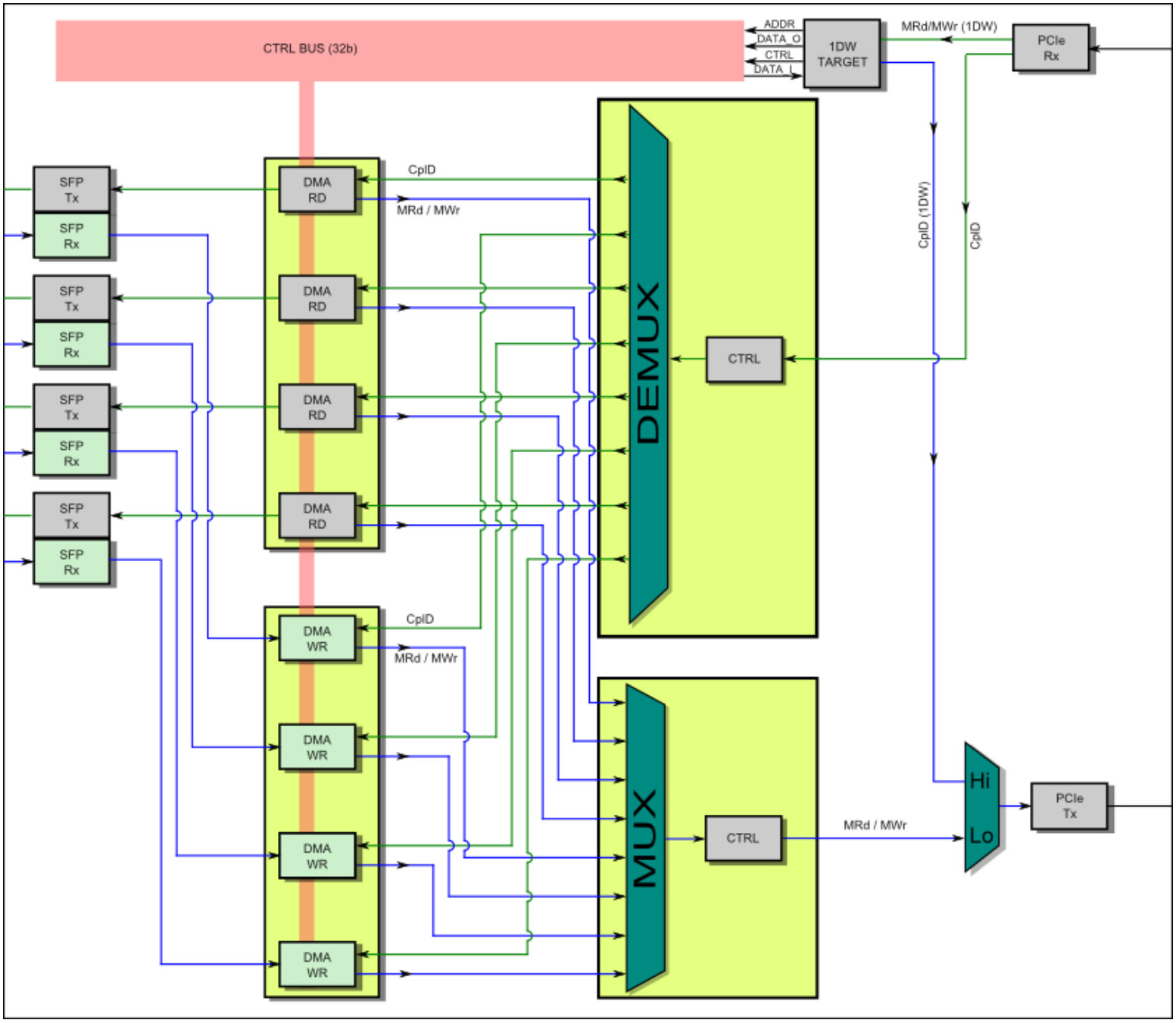
- Many projects @CERN looks similar to this …
- FPGA device(s) inside data treatment pipeline
- Data IN (could be SFP)
- Data OUT (could be PCIe, or Ethernet or something else)
- I was involved (2006-2009) in CMS/ECAL DCC project
  - Did some basic SV+AVM





An experiment

- VHDL + Verification Environment + Project Management/Build/Testing system
- VHDL
  - PCIe DMA engine for FPGA
- Verification Environment
  - SV + UVM
- Wrapped with …
  - Complete software (and hardware) development environment with Contiguous Integration (CI) setup.


CTRL BUS (32b)
ADDR
DATA_O
CTRL
DATA_I
1DW TARGET
MRd/MWr (1DW)
PCIe Rx
SFP Tx
SFP Rx
DMA RD
CplD
MRd / MWr
DEMUX
CTRL
CplD (1DW)
CplD
DMA WR
MUX
Hi
Lo
PCIe Tx

Building blocks

- Application I/O
  - In my example, I use a simple interface (DATA+DAV+SYNC)
  - Could carry some protocol like Ethernet (IP,UDP,TCP)
- Interconnect
- System I/O
  - Example uses PCIe, but could be exchanged to Ethernet (1/10/40 Gb)

Some assumptions

- Stay vendor independent:
  - No QSYS
  - No EDK/IP Integrator

Layered Application I/O

- SFP transceiver used to transmit data
- Simple signaling:
  - DATA + DAV (packet length 1-512 Bytes)
  - SYNC/TRIGG (pulse + packet length 0-1 Bytes)
- TB
  - Random: content, length, gap between packets, presence of SYNC/TRIGG
  - Predefined data patterns to help debugging
- Simulation
  - Slower Sim: XCVR PHY + IP included
  - Faster Sim: No PHY/IP, SV transaction model

Interconnect

- Internal Interconnect:
  - Much simpler than Avalon-ST/AXI Streaming
  - Independent Upstream + Downstream patch
  - Each direction with DATA+CTRL (example 128b+110b)
  - Cut-through or Store-and-forward operation mode
  - Split bus protocol (MRd-CplD/Cpl, MWr, Msg [similar to PCIe])
  - Channels switched with Deficit Weighted Round Robin (Insolvency enabled)
- TB
  - White box verification with assertions (signaling) and scoreboards (packet level)
- SIM
  - Only FIFOs components are vendor dependent

PCI express block

- Device Hard IP
  - Altera Gen1 x8 (i.e. Avalon-ST 128b@125 MHz if)
  - In the future Gen2 and Gen3 (256b@250MHz)
- Split-bus protocol (many transactions in flight)
- TB
  - Emulation of PC/ARM platform (buffers allocation)
    - Involves getting many SignalTap snapshots.
  - Split-bus packet (MRd -> CplD/Cpl) has to be handled with “real” life latency and bandwidth.
- SIM
  - Slow sim: Altera PCIe Verilog BFM wrapped with SVM+UVM
  - Fast sim: TLP BFM (limited functionality/does not inject all the “crap” seen with vendor BFM)

TB

- Interfaces:
  - SFP
  - PCIe
  - I2C (sensors: I/U/T)
- SV + UVM
  - VIP 1 – SFP RX
  - VIP 2 – SFP TX
  - VIP 3 – PCIe RX + TX
  - VIP 4 – I2C

Basic verification plan (2)

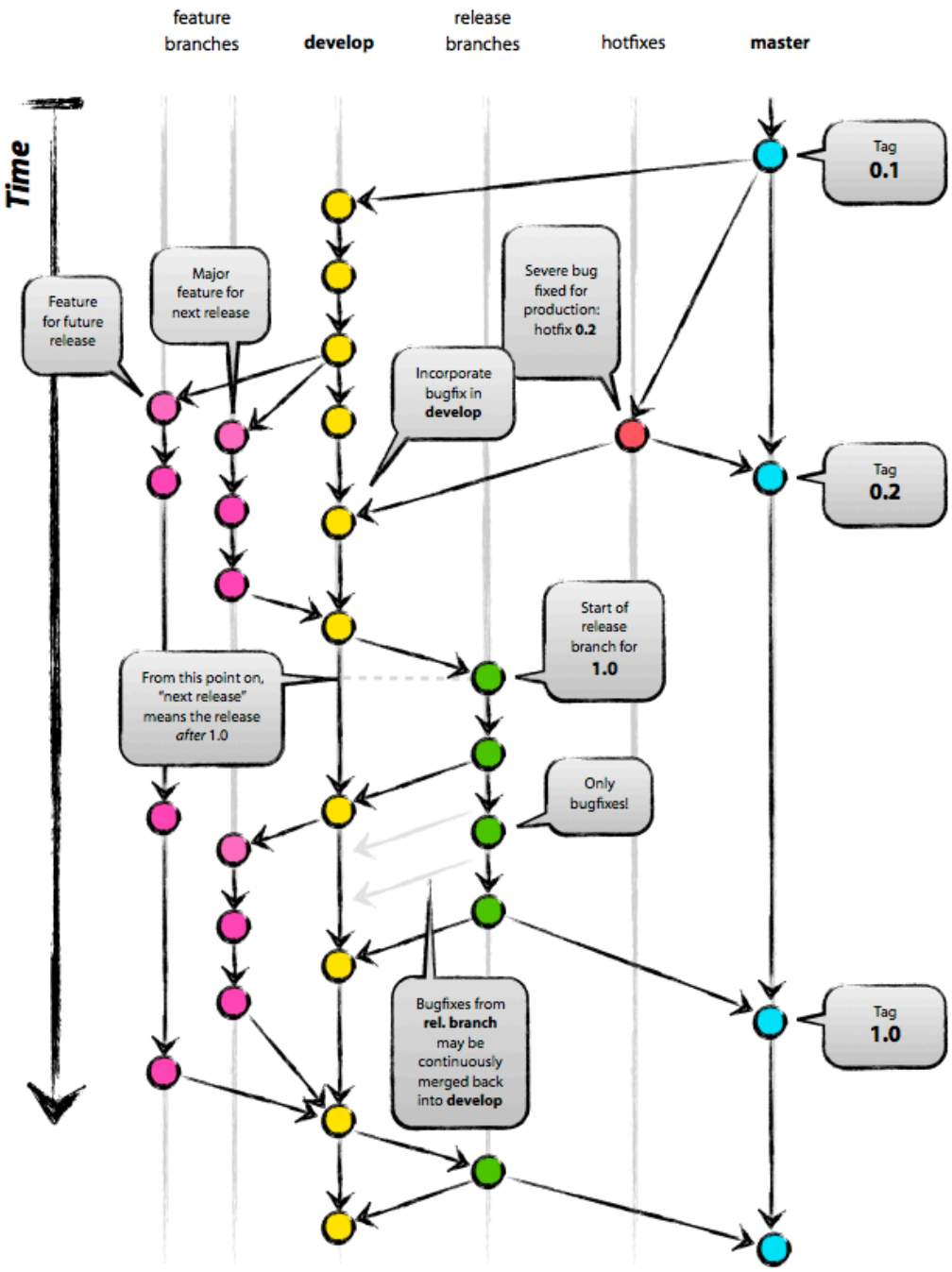
- SFP TX:
  - DMA CFG -> D:MRd -> CplD
    -> D:MWr
  - DMA RUN -> U:MRd -> CplD (DESC GET)
    -> U:MRd -> CplD (DATA GET)
  - Forward data (CplD) to SFP TX channel
  - DMA DONE -> U:MWr (DESC DONE)
    -> U:Msg (INT)
- How to distinguish between

Basic verification plan (2)

- SFP RX:
  - DMA CFG -> D:MRd -> CplD
    -> D:MWr
  - DMA RUN -> U:MRd -> CplD (DESC GET)
    Receive data over SFP RX channel
  - DMA RUN -> U:MWr (DATA PUT)
  - DMA DONE -> U:MWr (DESC DONE)
    -> U:Msg (INT)

Git+GitLab

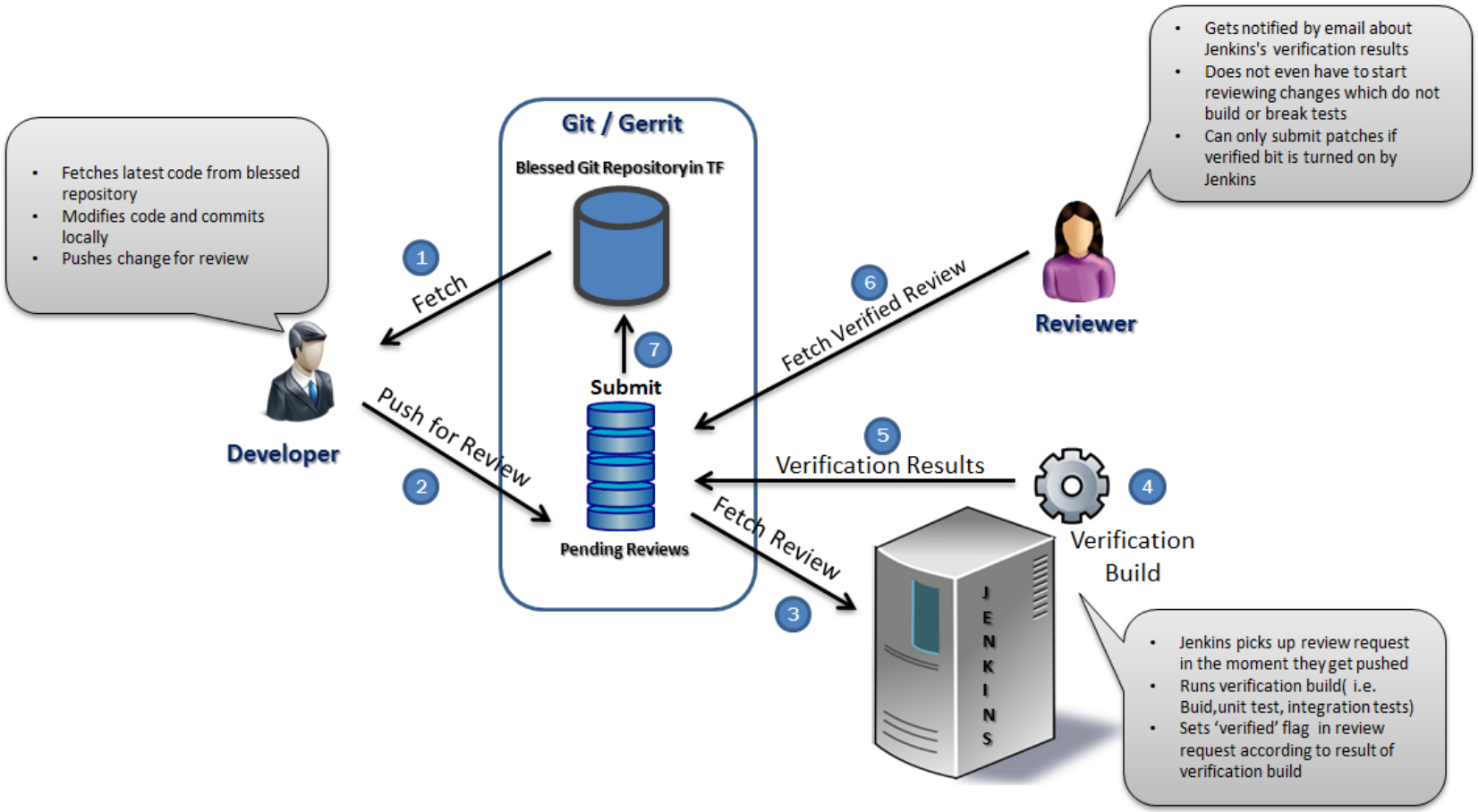
- Git
  - Distributed revision control
  - Lightweight branching and merging (compared to SVN)
  - Strong support for nonlinear development flow
- GitLab
  - Self hosted Git management service (your very own private GitHub)
  - Source file viewing
  - Forking projects
  - Issue tracking/Wiki/Code snippets exchange


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

Jenkins

- Continuous Integration
  - Regression testing
  - Automatic feature verification (bottom line: your TB has to be self-checking, waveforms only for debugging)
  - Timing analysis (Quartus/Vivado)
  - Bitstream builds
  - Many plugins (mainly around software development)

Git / Gerrit Work Flow with Jenkins Continuous Integration



Jenkins (xUNIT) + UVM

- How to use SV+UVM with Jenkins ?
  - Good start is to use JUNIT capability available in Jenkins – you need XML log file which follows JUNIT format
  - Easiest way is to replace UVM reporting server with your own one which generates XML log (already done by Verilab, source code published)
  - Write XSL schema to transform your custom XML into JUNIT XML format.
  - Your own XML + XSL can be feed into Jenkins' xUnit plugin which will feed JUNIT plugin automatically.
  - From then on, you can use all reporting available in Jenkins (error reports, trends analysis, etc.)

Summary

- What is working
  - Git + GitLab server
  - Jenkins + UVM XML reporter + QuestaSim
- Next steps

Learning resources

- Verification Academy (courses, videos, forum)
- Cadence UVM Book
- SystemVerilog for Verification, 3rd
- Verilab DVcon 2012 paper
- CIENA SNUG 2013 Jenkins paper